Challenges of Model-Driven Modernization: An Agile Perspective

Stavros Stavru$^1$, Iva Krasteva$^2$ and Sylvia Ilieva$^{3,1}$

$^1$Faculty of Mathematics and Informatics, Sofia University St. Kliment Ochridski, James Bouchier Str. 5, Sofia, Bulgaria
$^2$Rila Solutions, Acad. G. Bonchev Str., bl. 27, Sofia, Bulgaria
$^3$IICT-BAS, Acad. G. Bonchev Str., bl. 2, Sofia, Bulgaria

stavross@fmi.uni-sofia.bg, ivak@rila.bg, sylvia@acad.bg

Keywords: Agile Software Development, Software Modernization, Software Migration, Model-Driven Development, Model-Driven Modernization, Systematic Review.

Abstract: Software organizations are nowadays facing increased demand for modernizing their legacy software systems using up-to-date technologies. The combination of Model-Driven Development and delivery models like Cloud and Software as a Service have become a very promising approach for software modernization that possesses a lot of advantages, including great deal of automation and reuse of system functionality. However, the use of such new and immature technologies is very challenging and requires a comprehensive methodology for their seamless application within the software modernization projects. When developing such methodology, questions on whether agile methods and techniques should be incorporated and what could be the benefits and implications from that become of particular interest. To help answering these questions, the paper evaluates the potential of agile methods and techniques to address the challenges of Model-Driven Modernization. The challenges are extracted through a systematic review of the existing body of literature on Model-Driven Development and Software Modernization, and the evaluation is conducted through the Delphi technique. As a result, a ranked list of applicable agile techniques is proposed and suggestions for their use in Model-Driven Modernization are made.

1 INTRODUCTION

Software organizations are continuously pressured by their dynamic and highly competitive environments to modernize. As rebuilding legacy systems from scratch could require a huge investment in time and efforts, new and more lightweight modernization approaches are needed. With the growing popularity of Service-Oriented Architecture, the reuse of a legacy system by exposing its functionality through services was identified as a feasible and very promising modernization approach. This resulted in various service-oriented methodologies which were partially or specifically focused on software reuse. Such methodologies include the Service Migration and Reuse Technique developed by Carnegie Mellon and Software Engineering Institute (Lewis et al. 2005), Service-Oriented Analysis and Design and Service-Oriented Modeling and Architecture proposed by IBM (Arsanjani 2004, Zimmermann et al. 2004), Service Architecture Engineering (Butler 2007), Service Oriented Development of Applications published by Gartner (Plummer 2001) and the SOA Migration Framework (Razavian and Lago 2009). Although Service-Oriented approaches were notable improvement in terms of time and effort needed to modernize, there were still many issues, mostly concerned with the lack of in-depth knowledge on the migrated system and its obsolete and heterogeneous technologies and platforms. To overcome these challenges, model-driven techniques and tools were also incorporated. By turning models into primary artefacts of the modernization process, researchers and practitioners were expecting to automate or semi-automate many of the cumbersome activities, including the extraction of knowledge from the legacy system, the development of component architectures and detailed platform specific design, and even the generation of specific platform implementations. However, model-driven methodologies for software modernization are still scarce. They were mostly part of a small number of industrial and research projects, such as MOMOCS (www.momocs.org), SOAMG (www.soamig.de), MOOSE (www.moosetechnology.org), MoDisco (www.eclipse.org/modisco), and various initiatives.
of the Object Management Group, including Architecture-Driven Modernization (ADM) and Model Driven Architecture (MDA). However, they provide only partial support for Model-Driven Modernization (MDM) or are in their earliest stage.

REMICS (REuse and Migration of legacy systems to Interoperable Cloud Services) is an EU FP7 research project with the objective of supporting the modernization of legacy systems to service clouds by providing a model-driven methodology and tools. REMICS proposes to improve existing model-driven approaches (such as OMG’s MDA and ADM) and extend them when needed to provide a holistic view to migration that covers the whole process with a methodology, tools, languages and transformations (Mohagheghi et al. 2010).

Agile methods have been successfully applied in software industry in the recent decade (Dyba and Dingsoyr 2009). During that period their adoption has span from small development projects with low risk and criticality to large distributed projects in critical application domains such as banking and automotive industry. Recently, a number of studies on the applicability of agile methods and techniques in the areas of Model-Driven Development, Software Modernization, Service-Oriented Architecture and Cloud Computing have been published (Qumer and Henderson-Sellers 2007, Abbattista et al. 2009, Picek 2009, Prakash 2010, Matinnejad 2010). This paper further enlarges the existing body of research by answering the following research questions:

RQ1: Which agile techniques could add value to a given MDM methodology on the Service Clouds?
RQ2: How do agile techniques address the challenges of MDM on the Service Clouds?

In order to answer the above questions we propose a systematic approach for reviewing the challenges for MDM on the Service Clouds and an evaluation of various agile techniques in terms of their potential to address these challenges. The challenges that were reviewed are extracted from four fields related to MDM and Service Clouds, namely Model-Driven Development, Software Modernization, Service-Oriented Architecture and Cloud Computing. In order to provide particular focus for this study, we present only the first two fields in this paper, while the challenges from the other two fields are presented in (Stavru et al. 2012). The three main contributions of this paper are: (1) the identification and systematization of the challenges for MDM; (2) the discussion on the possible implications of these challenges on the incorporation of agile methods and techniques into the MDM process; and (3) the recommendation of agile techniques which could address the identified challenges.

The paper is organised as follows: Section 2 describes the methodology used for conducting literature review and evaluating agile techniques; Section 3 presents the challenges from the fields of Model-Driven Development and Software Modernization, extracted by the review process and relevant for the MDM on the Service Clouds; Section 4 discusses the results of the evaluation of agile techniques and their potential to address the identified challenges; and Section 5 concludes the paper and outlines directions for future research.

2 METHODOLOGY

The methodology used to assess Agile Software Development in the context of MDM consists of two consecutive phases. During the first phase, the challenges of both fields of Model-Driven Development and Software Modernization were extracted, analyzed and synthesized through a systematic literature review (Brereton et al. 2007). Then, in the second phase, using the Delphi method (Helmer and Helmer-Hirschberg 1983), various agile techniques (taken from XP and Scrum) were evaluated by a panel of experts for their potential to address the identified challenges. The methodology and its phases are thoroughly described in the next paragraphs.

2.1 Review

Systematic review provides an analytical review scheme, which is necessary for evaluating the contribution of a given body of literature. It employs an objective, transparent and reproducible procedure for the identification, appraisal, selection and synthesis of studies highly relevant to specific research questions and thus improves the quality of the review process and its outcome (Brereton et al. 2007).

The systematic review, presented in this paper, was conducted following the approach proposed by Kitchenham (Kitchenham 2004), taking into account also her guidelines on performing systematic reviews in software engineering (Kitchenham 2007). Given the review objective for providing a comprehensive overview and a conceptual, rather than an empirical, consolidation of the literature in regard to the challenges of MDM,
the data analysis was limited to descriptive, rather than statistical (meta-analysis) methods and data synthesis was conducted using qualitative methods (or meta-ethnographic methods in particular). The report of the conducted review will be briefly summarized as it involves an extensive research which cannot be presented within the limits of this paper.

Articles were eligible for inclusion in the review based on their relevance to the review objectives, which are: (1) they describe the current state of research and practice in Model-Driven Development and Software Modernization; and (2) they identify and discuss different challenges these areas poses to both academia and industry. The relevance was evaluated by reviewing the abstracts of the articles and grading them as either relevant or irrelevant. The inclusion was also restricted by the type of the study, including only review articles and/or Software Modernization; and (2) they have an abstract or the abstract is not available from the included electronic database; (2) the access to the full text of the article is restricted; and (3) the full text of the article is not available in English.

The search strategy included both journals and conference papers, and was limited to the Scopus electronic database. Scopus is the largest abstract and citation database of research literature and quality web sources, which ensured the coverage of nearly 18,000 titles from more than 5,000 publishers. The titles of both journals and conference papers were searched using the following search terms:

1. Model-Driven Development - (“Model-Driven” AND (Challenges OR Review OR Landscape OR Roadmap OR “State of”));
2. Software Modernization - (“Software” AND (Modernization OR Migration OR Legacy OR Transformation) AND (Challenges OR Review OR Landscape OR Roadmap OR “State of”).

Applying the search strategy resulted in an initial pool of 43 articles, 35 articles for Model-Driven Development and 9 for Software Modernization. Some additional articles, not covered by the search strategy, were also included as being recommended by the research community. Thus, by using the inclusion and exclusion criteria the initial pool of articles was limited to 26 articles. Their full texts were thoroughly examined in order to extract the challenges of Model-Driven Development and Software Modernization, which are presented in the subsequent sections.

2.2 Evaluation

The methodology used to evaluate various agile techniques in terms of their potential to address the challenges of MDM was the Delphi technique. This technique is frequently used for eliciting consensus from within a group of experts and has many advantages over other methods of using panel decision making (Helmer and Helmer-Hirschberg 1983). Various researchers have found that one of the major advantages of using it as a group response is that consensus will emerge with one representative advantages of using it as a group response is that consensus will emerge with one representative opinion from the experts (Linstone and Turoff 1975, Helmer and Helmer-Hirschberg 1983). Other advantages include its simplicity, anonymity, controlled feedback from the interaction, etc. (Yousuf 2007). Some limitations include that judgments are derived from the subjective opinions of experts and may not be representative, it requires adequate time and participant commitment, its validity extremely depends on the expertise and experience of the panellists, etc. (Yousuf 2007). However, Linstone (Linstone and Turoff 1975) recommends the Delphi technique when the examined issue does not allow the use of analytical techniques but can benefit from the subjective judgments on a collective basis, which is our case.

The process followed was the one proposed by Pfeiffer (Pfeiffer 1968) and included three subsequent phases. During the first phase (recommendation phase), a questionnaire was sent to a panel of experts (with an average of 9 years of both academic and industrial experience in Agile Software Development), asking them to review the list of challenges extracted by the review process and make subjective judgment and recommendations on which agile techniques (from Scrum and XP) could be used to address these challenges. From each expert a list of agile techniques was obtained. During the second phase (evaluation phase), a consolidated list of agile techniques was obtained.

Applying the search strategy resulted in an initial pool of 43 articles, 35 articles for Model-Driven Development and 9 for Software Modernization. Some additional articles, not covered by the search strategy, were also included as being recommended by the research community. Thus, by using the inclusion and exclusion criteria the initial pool of articles was limited to 26 articles. Their full texts were thoroughly examined in order to extract the challenges of Model-Driven Development and Software Modernization, which are presented in the subsequent sections.

2.2 Evaluation

The methodology used to evaluate various agile techniques in terms of their potential to address the challenges of MDM was the Delphi technique. This technique is frequently used for eliciting consensus from within a group of experts and has many advantages over other methods of using panel decision making (Helmer and Helmer-Hirschberg 1983). Various researchers have found that one of the major advantages of using it as a group response is that consensus will emerge with one representative opinion from the experts (Linstone and Turoff 1975, Helmer and Helmer-Hirschberg 1983). Other advantages include its simplicity, anonymity, controlled feedback from the interaction, etc. (Yousuf 2007). Some limitations include that judgments are derived from the subjective opinions of experts and may not be representative, it requires adequate time and participant commitment, its validity extremely depends on the expertise and experience of the panellists, etc. (Yousuf 2007). However, Linstone (Linstone and Turoff 1975) recommends the Delphi technique when the examined issue does not allow the use of analytical techniques but can benefit from the subjective judgments on a collective basis, which is our case.

The process followed was the one proposed by Pfeiffer (Pfeiffer 1968) and included three subsequent phases. During the first phase (recommendation phase), a questionnaire was sent to a panel of experts (with an average of 9 years of both academic and industrial experience in Agile Software Development), asking them to review the list of challenges extracted by the review process and make subjective judgment and recommendations on which agile techniques (from Scrum and XP) could be used to address these challenges. From each expert a list of agile techniques was obtained. During the second phase (evaluation phase), a consolidated list of agile techniques was obtained.
order to discuss big differences in ratings. It was decided that an agile technique would be considered as having the potential to address a specific challenge only if the final level of agreement from each expert is either “Agree” or “Strongly Agree”. After a number of iterations for clarifications and argumentation, a consensus was gained, resulting in a sorted list of agile techniques and the challenges they could address.

3 CHALLENGES OF MDM

The challenges identified by the review process were sorted into two categories:

(1) Organizational challenges – These are process- and people-oriented challenges from all levels of the organization, including: (1) strategic challenges (e.g. organizational restructuring and evaluation of business context); (2) managerial challenges (e.g. competence acquisition and lack of commitment and support); and (3) operational challenges (e.g. etc. lack of process models, interoperable tools and integrated development environments);

(2) Technical challenges – These are product- and technology-oriented challenges related to the specification, design, implementation and verification of the modernized system.

As the focus of this study was on MDM, we expected that not all of the challenges discussed by the reviewed articles would be relevant. For that reason, we limited the extraction of challenges to only these challenges which are applicable either in general or in the context of MDM, excluding challenges which are applicable only in very specific contexts (e.g. embedded and safety-critical systems).

3.1 Challenges of Model-Driven Development

Total of 18 reviews were thoroughly examined in order to extract the challenges relevant to Model-Driven Development (MDD). As many challenges were found, they were further consolidated into total of 12 challenges, 7 of which were organizational and 5 were technical challenges. A summary of these challenges, together with their references, is presented in Table 1.

Table 1: Challenges of Model-Driven Development.

<table>
<thead>
<tr>
<th>#</th>
<th>Challenge</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td>Lack of process models</td>
</tr>
<tr>
<td>O2</td>
<td>Acquisition of competencies and expertise</td>
</tr>
<tr>
<td>O3</td>
<td>Restructuring of software development team</td>
</tr>
<tr>
<td>O4</td>
<td>Restructuring of software development lifecycle</td>
</tr>
<tr>
<td>O5</td>
<td>Reliance on high level models</td>
</tr>
<tr>
<td>O6</td>
<td>Emphasizing technology rather than humans</td>
</tr>
<tr>
<td>Technical challenges</td>
<td>Capability/ability</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>T1 Management of models</td>
<td>While models at varying levels of abstractions are created, evolved, analyzed and transformed, many integration (e.g. versioning, merging, etc.), consistency (between different models and levels of abstractions, interrelations and dependencies, etc.) and scalability (e.g. scaling beyond a few tens of thousands of model elements per model) issues arise (Hailpern and Tarr 2006, France and Rumpe 2007, Streitferdt et al. 2008, Teppola et al. 2009, Kolovos et al. 2009, Mohagheghi et al. 2009, Rivera et al. 2009, Hailpern and Tarr 2006, Straeten et al. 2009, Kolovos et al. 2009, Tajali et al. 2011).</td>
</tr>
<tr>
<td>T2 Transformation of models</td>
<td>Synchronization transformation technologies are required to “ripple” the results of transformations to related views or to address the round-trip problems between the models (incl. generated code), together with new procedures for verifying the correctness of these model transformations, etc. (Hailpern and Tarr 2006, France and Rumpe 2007, Streitferdt et al. 2008, Teppola et al. 2009, Kolovos et al. 2009, Rivera et al. 2009, Tajali et al. 2011, Vangheluwe 2011).</td>
</tr>
<tr>
<td>T3 Design of the modeling languages</td>
<td>The design of the modeling language is identified as one of the major aspects related to modularity in modeling. But how the modeling language should be defined remains an open research topic, including issues as how to provide support for creating and manipulating problem-level abstractions as first-class modeling elements in a language (the abstraction challenge), what aspects of the semantics of the modeling language need to be formalized and how this should be done, etc. (Hailpern and Tarr 2006, France and Rumpe 2007, Conny and Paige 2007, Teppola et al. 2009, Straeten et al. 2009, Kolovos et al. 2009, Vangheluwe 2011).</td>
</tr>
<tr>
<td>T5 Integration of legacy / handcrafted code</td>
<td>In MDD there could be the case that software engineers, after performing model-to-code transformations, have to integrate generated code with handcrafted or legacy code. This could result in refactoring of the generated and foreign code, as well as writing glue code (France and Rumpe 2007, Streitferdt et al. 2008, Kolovos et al. 2009, Mohagheghi et al. 2009).</td>
</tr>
</tbody>
</table>

The descriptive analysis of the extracted challenges revealed that organizational and technical challenges were almost equality considered within the reviewed literature. Organizational challenges were examined by 89% of the reviewed articles with an average of 5.4 articles per organizational challenge, while the same numbers for technical challenges were 75% and 7.4 respectively. The most cited organizational challenge was the lack of mature tools, integrated development environments and off-the-shelf infrastructure (O7) with total of 11 citations (or 61% of the reviewed articles), followed by competence acquisition (O2) and the reliance on high level models (O5) with 10 (56%) and 8 (44%) citations respectively. In terms of technical challenges, model transformations (T2) have been cited the most (by total of 11 articles or 61% of all articles), followed by model management (T1) and language challenges (T3) with 9 (50%) and 7 (39%) citations respectively. Further analysis revealed three reasons for observing the extracted organizational challenges: (1) the early adoption stage of Model-Driven Development (O1, O2, O7); (2) its technology intensive nature (O2, O6); and (3) the significant organizational change needed when moving from traditional to model-driven software development (O2, O3, O4, O5). The reasons for observing the technical challenges were mostly associated with the use of new and innovative technologies, where high level abstractions are used for specifying, designing, implementing and verifying complex software systems.
The challenges in Table 1 have various implications on the incorporation of agile methods and techniques into the MDM process. Model-Driven Development, by emphasizing technology rather than humans (O6), shifts the focus on integrated development environments, tools and technologies and sees them as the primary factor for success. This contradicts with the agile philosophy, which postulates that people are the most valuable asset of the organization and human/social aspects are the key for the successful software development. Another implication is that there are many prerequisites before models can be turned into fully operational (or working) software system, even if the software system provides only limited (but still valuable) functionality. Example of such prerequisites, which are requiring considerable time and effort, are: (1) the definition of the problem domain (O5); (2) the specification of the domain language, transformation language, etc. (T3); (3) the provision of infrastructure (such as meta-models, model transformations), configuration tools and build processes (O7); (4) the integration of legacy and / or handcrafted code (T5); and many others. All these prerequisites pose some limitations on the extent to which agile methods and techniques could be incorporated. In Agile Software Development, working software is the only recognized measure of progress and the rapid delivery is the ultimate goal, so not having working software in the early stage of the development lifecycle could be problematic. Also, having so many prerequisites could threaten the effectiveness of using short increments (2 - 4 weeks) and the possibility of delivering potentially shippable products. This could result in reduced customer value (e.g. through delayed time to market) and untimely customer feedback (e.g. receiving the feedback too late in the development lifecycle). The challenges related to the management and transformation of models (T1, T2) could significantly hinder organization's ability to respond to change due to the extensive and time-consuming efforts needed for securing model integration, consistency, scalability, transformation, etc. T1 and T2, together with ensuring model quality (T4), could also result in redundant documentation and architecture, which could further delay the delivery of working software. Probably the biggest implication for incorporating agile methods and techniques is that Model-Driven Development emphasize on models rather than on coding and testing (O5), and requires restructuring of the traditional software development teams and lifecycles (O3, O4). This could further hamper the agile implementation into MDM because many of the existing agile techniques might need significant modifications (e.g. pair programming to be adapted as pair modeling). Such possible modifications were already discussed in the work of Zhang and Patel (Zhang and Patel 2011).

Although there are many implications for incorporating agile methods and techniques into Model-Driven Development, they seem to share some common values. For example, using models as the primary artifacts in the development process is promising to increase the customer value (through better understanding of the problem domain from all parties, incl. engineers; closer customer collaboration and effective requirements elicitation and prioritization; etc.) and the organization's ability to respond to change (by changing only the high level models and automatically distributing the changes to the implementation / testing code). However, in Model-Driven Development there is still no special attention on individuals and interactions, customer collaboration and working software – values central to Agile Software Development.

### 3.2 Challenges of Software Modernization

A total of 8 reviews in the field of Software Modernization were examined. The extracted challenges were further consolidated into 11 challenges, including 7 organizational challenges and 4 technical. They are shown in Table 2.

<table>
<thead>
<tr>
<th>#</th>
<th>Challenge</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1</td>
<td><strong>Definition of business context</strong></td>
</tr>
<tr>
<td></td>
<td>Software modernization (as a heavy initiative) to be successful needs to be considered in regard to the specific business / organizational context and aligned with the existing business goals and strategies, project constraints as time / budget, organizational stakeholders, etc. Thus evaluating the feasibility of the modernization initiative (e.g. through pilot modernizations, prototypes, etc.) should be an inevitable part of the modernization process (Lewis et al. 2005, Mohagheghi and Sæther 2011).</td>
</tr>
<tr>
<td>O2</td>
<td><strong>Lack of business commitment</strong></td>
</tr>
<tr>
<td></td>
<td>Software modernization success also depends on how well the modernization efforts are justified in terms of business value and the extent to which these efforts are supported by all organizational stakeholders, including customers and</td>
</tr>
</tbody>
</table>
management. This requires additional efforts in order to motivate the various stakeholders and gain their commitment (Lewis et al. 2005, Teppe 2009).

### O3 Resistance to change

Legacy systems are often critical for business (these systems are matured, heavily used, and constitute massive corporate assets), so stakeholders (incl. business people, customers, etc.) could be sensitive about the modernization process and might resist too many changes due to higher risk (Chowdhury and Iqbal 2004, Lewis et al. 2005, Al-Azzoni et al. 2011).

### O4 Acquisition of competencies and expertise

The software users must be re-trained and equipped to use and understand the new applications and platforms effectively. The same holds for the whole software development team, which has to be re-trained to use the modernization tools as well as the new technologies, languages, platforms, etc. that are involved (Teppe 2009).

### O5 Increased risk due to new technologies

New technologies should be prior evaluated for both business and technical feasibility as they might have tremendous impact on the successful modernization of the legacy systems (Lewis et al. 2005).

### O6 Lack of in-depth legacy knowledge

It might be the case that there's a lack of in-depth knowledge regarding the functional / non-functional aspects and requirements for a given legacy system (due to incomplete, inadequate, out-of-date or missing documentation, the people who developed the system had left the organization, etc.), while the target system is expected to reflect the source system as it is (Chowdhury and Iqbal 2004, Kvam et al. 2005, Al-Azzoni et al. 2011).

### O7 Lack of support

The business people and customers might provide limited or no assistance during the modernization process. They might not have the time or possibility to re-evaluate their business needs in order to re-evaluate the source system and extend its functionality. Also they might point the source systems as the ultimate source for requirements (Lewis et al. 2005, Teppe 2009).

### Technical challenges

#### T1 Extracting business and technical knowledge from legacy systems

The need to extract business and technical knowledge could be problematic due to incomplete, inadequate, out-of-date architectural and design documentation, low quality of source code, lack of unit / acceptance tests, etc. This requires additional efforts (manual or semi-automated) and could become a time consuming activity (Chia-Chu and Bayrak 2006).

#### T2 Ensuring behavioural equivalence between source and target systems

Additional efforts are also needed in order to secure the behavioural equivalence (in terms of functionality) between the source and target systems (Lewis et al. 2005, Chia-Chu and Bayrak 2006, Torchiano et al. 2008).

#### T3 Co-existence of source and target systems

Additional issues arise when the source and target systems should co-exist not only during the modernization process but also when it is finalized (e.g. integration issues, replication of efforts, data, utilization of resources, etc.).

#### T4 Overcome obsolete and/or heterogeneous technologies (both software and hardware)

It could be the case that the legacy system is based on obsolete and heterogeneous technologies and platforms, which makes harder the modernization process to new technologies / platforms due to limited possibilities for reuse (in terms of design, architecture, implementation details, test suits and etc.). Example is migration from procedural programming paradigm to object-oriented programming paradigm, or migration from a mixture of COBOL/Delphi.NET/C# based system to pure Java based system and etc.

The descriptive analysis of the extracted challenges revealed that organizational challenges are prevailing within the reviewed literature. Organizational challenges were examined by 75% of the reviewed articles with an average of 2 articles per organizational challenge, while the same numbers for technical challenges were 38% and 1 respectively. The most cited organizational challenges were the resistance to change (O3) and the lack of in-depth knowledge (O6) with total of 3 citations each (38%), followed by the definition of business context (O1) and the lack of business commitment (O2) with 2 citations (25%) respectively. In terms of technical challenges, ensuring behavioural equivalence (T2) have been cited the most, by total of 3 articles (38%), followed by extracting business and technical knowledge from legacy systems (T1) with only one citation. The co-existence of source and target systems (T3) and the overcoming obsolete and/or heterogeneous technologies (T4) were not cited in any of the reviewed articles. They were additionally included by the authors, based on their own experience with software modernization in real industrial settings. The dominance of organizational values within the
reviewed literature could be explained by the significant organizational change, required by the modernization process, which affects crucial business assets of the organization. So how this change will be introduced and managed within the organization becomes an arduous task. The technical challenges on the other hand are context specific (depend on the specific technology / platform / programming language / etc. of the source and target systems), although there are some general technical challenges relevant to any software modernization initiative (Table 2).

Software Modernization and its challenges further affect the way agile methods and techniques could be incorporated into the MDM process. Challenges as lack of business commitment (O2) and support (O7) could negatively impact customer collaboration, which is one of the key success factors for the implementation of Agile Software Development. The co-existence of source and target systems (T3) could reduce the customer value (in terms of new functionality, improved quality, etc.) as it could shift the focus from enhancing the modernized system to keeping both source and target systems aligned and synchronized. The increased risk (e.g. failure due to inappropriate selection of technologies and process models) (O5) and stakeholders’ sensitivity to the modernization process (e.g. crucial business assets are being changed), could lead to significant change resistance and organizational rigidity. This could limit organization’s ability to respond to change.

Although there are some implications for incorporating agile methods and techniques in the context of Software Modernization, our analysis revealed that they are no significant obstacles for scaling Agile Software Development for modernizing complex software systems.

4 RESULTS

The present section discusses the results of the evaluation of agile techniques based on the Delphi method. The results are shown in Table 3, where the techniques are sorted by the total number of challenges they are expected to address (shown in brackets next to the technique’s name).

The agile techniques with the highest potential to address the challenges of MDM (total of 18 challenges) were Small Releases (from XP), Sprints (from Scrum) and Cross-Functional Teams (Table 3). One of the arguments for Small Releases and Sprints was (1) receiving rapid feedback, including feedback from the process (e.g. the adequacy of team roles and responsibilities, and the development lifecycle), from the product (e.g. the effectiveness of the problem domain, modeling languages, model transformations, etc. and the quality of the models) and from the people (e.g. lack of commitment and support). Another argument was (2) increasing the organizational responsiveness to change by allowing changes to happen in each subsequent increment (e.g. refinement of the problem domain, modeling languages, model transformations, etc. and changes in the infrastructure, build processes, tools and integrated environments, etc.). Among the other arguments for incorporating Small Releases and Sprints were: (3) gaining commitment and support through frequent communication, increased visibility and traceability, etc.; (4) effective competency acquisition through learning by doing; (5) reducing risk through early detection of potential issues; (6) early delivery of customer value; and etc.

Cross-Functional Teams (from Scrum) was also highly recommended by the experts. Among the arguments for using Cross-Functional Teams, together with Whole Team and Pair Programming (from XP), were: (1) the effective acquisition of competencies and expertise (e.g. through daily knowledge transfer and direct interaction); (2) reduced risk (e.g. through homogeneous distribution of knowledge and expertise); (3) emphasis on human/social aspects (e.g. through empowering the team, building trust and respect, and enhancing collaboration and interaction between individuals); (4) effective competency acquisition through learning by doing; (5) reducing complexity (e.g. all required knowledge and expertise are within the boundaries of the team); (6) increased responsiveness and support (e.g. through flawless communication and collaboration, and customer involvement); (6) securing non-functional concerns (e.g. the quality of models and the behavioural equivalence between the source and target systems; etc.

Planning Game (from XP) and Sprint Planning Meeting (from Scrum) also have strong potential for addressing the challenges of MDM (total of 15 challenges). Experts motivated their recommendations with: (1) active involvement of customers or their representatives in the development process; and (2) enhanced collaboration between customers and the development team. Bringing together the customer (or customer’s representative) and the development team before each iteration could also result in (3) increased customer value (e.g. through effective requirements elicitation and prioritization, refinement of the problem domain and mutual
Table 3: Agile techniques and the challenges they are expected to address.

<table>
<thead>
<tr>
<th>Agile Technique</th>
<th>MDD Challenges</th>
<th>SM Challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extreme Programming (XP)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Small Releases (18)</td>
<td>O2, O3, O4, O7, T1, T2, T3, T4, T5</td>
<td>O2, O3, O4, O5, O6, O7, T1, T2, T3</td>
</tr>
<tr>
<td>Whole Team (16)</td>
<td>O2, O3, O4, O5, O6, T4, T5</td>
<td>O1, O2, O3, O4, O5, O6, O7, T1, T2, T3</td>
</tr>
<tr>
<td>Planning Game (15)</td>
<td>O3, O4, O5, T2, T3</td>
<td>O1, O2, O3, O5, O6, O7, T1, T2, T3, T4</td>
</tr>
<tr>
<td>Pair Programming (13)</td>
<td>O2, O4, O5, O7, T2, T3, T4, T5</td>
<td>O4, O6, T1, T2, T4</td>
</tr>
<tr>
<td>Continuous Integration (11)</td>
<td>O4, O5, O6, O7, T1, T2, T4, T5</td>
<td>O3, T3, T4</td>
</tr>
<tr>
<td>Test-Driven Development (10)</td>
<td>O4, O5, O7, T4, T5</td>
<td>O3, O4, T1, T2, T3</td>
</tr>
<tr>
<td>Collective Code Ownership (7)</td>
<td>O3, O4, O5, T1, T2, T3</td>
<td>O6</td>
</tr>
<tr>
<td>System Metaphor (4)</td>
<td>O2, O4, O5</td>
<td>O4</td>
</tr>
<tr>
<td>Sustainable Pace (4)</td>
<td>O2, O6, O7</td>
<td>O4</td>
</tr>
<tr>
<td>Refactoring (3)</td>
<td>O4, O5</td>
<td>T4</td>
</tr>
<tr>
<td>Simple Design (2)</td>
<td>O4, O5</td>
<td>-</td>
</tr>
<tr>
<td>Coding Standards (2)</td>
<td>O4, O5</td>
<td>-</td>
</tr>
<tr>
<td>Scrum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sprint (18)</td>
<td>O2, O3, O4, O7, T1, T2, T3, T4, T5</td>
<td>O2, O3, O4, O5, O6, O7, T1, T2, T3</td>
</tr>
<tr>
<td>Cross-Functional Team (18)</td>
<td>O2, O3, O4, O5, O6, T1, T2, T3, T4, T5</td>
<td>O4, O5, O6, O7, T1, T2, T3, T4</td>
</tr>
<tr>
<td>Sprint Planning Meeting (15)</td>
<td>O3, O4, O5, T2, T3</td>
<td>O1, O2, O3, O5, O6, O7, T1, T2, T3, T4</td>
</tr>
<tr>
<td>Product Backlog (11)</td>
<td>T2, T3</td>
<td>O1, O2, O3, O5, O6, O7, T1, T2, T3</td>
</tr>
<tr>
<td>Spring Backlog (11)</td>
<td>T2, T3</td>
<td>O1, O2, O3, O5, O6, O7, T1, T2, T3</td>
</tr>
<tr>
<td>Product Owner (8)</td>
<td>O3, O4</td>
<td>O1, O2, O3, O6, O7, T1</td>
</tr>
<tr>
<td>Sprint Review Meeting (6)</td>
<td>-</td>
<td>O2, O3, O6, O7, T2, T3</td>
</tr>
<tr>
<td>Daily Scrum (4)</td>
<td>O4, T1, T5</td>
<td>T4</td>
</tr>
<tr>
<td>Scrum Master (4)</td>
<td>O3, O4</td>
<td>O5, O7</td>
</tr>
<tr>
<td>Scrum of Scrums (2)</td>
<td>O4, T1</td>
<td>-</td>
</tr>
<tr>
<td>Sprint Retrospective (2)</td>
<td>O2</td>
<td>O4</td>
</tr>
<tr>
<td>Sprint Burn Down Chart (0)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

understanding of the business context); (4) reduced risk for failure (e.g. due to collective estimations); (5) gaining support and commitment from stakeholders; (6) clarification of team roles and responsibilities; (7) early detection and escalation of concerns; etc. Other agile techniques, highly recommended by the experts were Continuous Integration and Test-Driven Development (from XP) and Product Backlog, Spring Backlog and Daily Scrum (from Scrum).

Based on the presented results and following the Pareto principle (80% of the effects come from 20% of the causes) (Pareto 1971), we recommend that an organization, undertaking MDM and interested in Agile Software Development, should start with small releases (or sprints), encourage cross-functional teams and incorporate planning meetings similar to either planning game or sprint planning meeting. This would guarantee minimum efforts for incorporating agile into the MDM process and maxim efficiency in terms of addressed challenges. Next, if the organization would like to further increase its agility, it might continue with Product / Sprint Backlogs, Continuous Integration and On-Site Customer. These techniques also have high potential to address the challenges of MDM. Finally, as almost all of the examined agile techniques could be beneficial for the MDM, an organization might also consider full implementation of either XP or Scrum (or a hybrid), as this will ensure cohesiveness and will allow the organization to take full advantage of these methods.

The results in Table 3 can be further used to sort out the techniques which have the potential to address a particular challenge. This might be useful when one or more challenges of MDM have greater impact on the project than others. Then, all the techniques recommended for that particular
challenge might be considered for introduction in the project.

5 LIMITATIONS

This study has its recognized limitations. Some of these limitations are coming from the review methodology used to extract the challenges of MDD and Software Modernization. Narrowing the search strategy to include only the titles of the published articles and limiting the publication databases to a single electronic database might have minimized the likelihood of capturing all relevant data and thus maximized the effects of publication bias. The risk of publication bias was further increased by excluding studies depending on their type of study, unavailability and language. However, by taking some additional actions (e.g. including studies recommended by the research community) and by rigorously following the procedure of the systematic review, the probability that the omitted research could have critically altered our findings and threaten their generalizability (or external validity) has been reduced.

The use of the Delphi method to evaluate agile techniques in terms of MDM possesses some limitations as well. Among the most critical limitations are that the evaluation was derived from the subjective opinions of experts, which may not be representative, and that the validity of the evaluation extremely depends on their expertise and experience. In order to mitigate these threads, the selection of the panellists was restricted to experts with more than 5 years of both academic and industrial experience in Agile Software Development, as well as proved knowledge and experience in MDD and Software Modernization.

6 CONCLUSIONS

This paper presented the challenges of Model-Driven Development and Software Modernization, which were extracted, analyzed, and synthesized through a systematic literature review. Then, using these challenges, the paper: (1) discussed the possible implications for incorporating agile methods and techniques into the MDM process; (2) evaluated various agile techniques (from XP and Scrum) for their potential to overcome the challenges of MDM; and (3) provided recommendations on which agile techniques are most applicable in the context of MDM and gave suggestions (following the Pareto principle) on how they should be incorporated into the MDM process. Although there were many implications for combining Agile Software Development and MDM, our final conclusion is that Agile Software Development and MDM are compatible and using various agile techniques could be beneficial for organizations that are approaching software modernization through model-driven development.

Adding our previous results in the fields of Service-Oriented Architecture and Cloud Computing (Stavru et al. 2012), our future work is proposing a comprehensive agile methodology for model-driven modernization of software systems with deployment in Service Cloud and the empirical evaluation of this methodology using the REMICS’s case studies.

ACKNOWLEDGEMENTS

The research in this paper was partially supported by FP7 project REMICS, contract No. 257793 and by National Scientific Fund, Bulgarian Ministry of Education and Science, Research Project agreement n. DO-02-182.

REFERENCES


Arsanjani, A. 2004. 'Service-oriented modeling and architecture'. *IBM developerWorks*.


Kitchenham, B. 2004. 'Procedures for performing systematic reviews.' Keele University and NICTA.


